MultiMasher: Providing Architectural Support and Visual Tools for Multi-Device Mashups

Maria Husmann, Michael Nebeling, Stefano Pongelli, and Moira C. Norrie

Department of Computer Science, ETH Zurich CH-8092 Zurich, Switzerland,
{husmann|nebeling|norrie}@inf.ethz.ch

Abstract. The vast majority of web applications still assume a single user on a single device and provide fairly limited means for interaction across multiple devices. In particular, developing applications for multi-device environments is a challenging task for which there is little tool support. We present the architecture and tools of MultiMasher, a system for the development of multi-device web applications based on the reuse of existing web sites created for single device usage. Web sites and devices can be mashed up and accessed by multiple users simultaneously, with our tools ensuring a consistent state across all devices. MultiMasher supports the composition of arbitrary elements from any web site, inter-widget communication across devices, and awareness of connected devices. We present both conceptual and technical evaluations of MultiMasher including a study on 50 popular web sites demonstrating high compatibility in terms of browsing, distribution and linking of web site components.

Keywords: web site mashups, distributed user interfaces, multi-device mashups

1 Introduction

Despite the widespread use of diverse computing devices in our daily lives, the overwhelming majority of applications are still built to be used on one device at a time. Creating applications that integrate multiple different devices is challenging and there is little tool support to cater for multi-device environments.

While there has been a great deal of research in the area of distributed user interfaces (DUI), most of it is either focused on building new applications from scratch [1, 2] or on distributing a single, existing application across multiple devices [3]. The potential of reusing and mixing multiple applications has not been a topic in the DUI community, although the benefits have been recognised in the mashup research community where a range of frameworks and tools for integrating existing components from different web sites into a new application have been proposed [4, 5]. However, the resulting applications are usually still developed for a single device.

In [6], we described a first prototype of MultiMasher with an initial set of visual tools designed for creating mashups and distributing them across multiple

connected devices. In this paper, we expand on the concepts of such multi-device mashups and present an updated and extended version of MultiMasher built on a new architecture. In particular, we make the following contributions.

- We introduce and illustrate *concepts* for multi-device mashups in Sect. 3.
- The updated MultiMasher client provides new *visual tools* which are described in Sect. 4. It runs in any modern browser and does not require browser extensions or plugins. It supports the selection of elements from arbitrary web sites and the composition of multi-device mashups in a direct manipulation interface. The selected elements can be arranged in mashups through drag-and-drop operations and new interactions may be defined between them. MultiMasher provides an overview of all connected devices and elements can also be moved easily between devices via drag-and-drop.
- A new *architecture* is presented in Sect. 5 which eliminates several limitations of the first prototype and enables additional functionalities.

In addition, we present both *conceptual and technical evaluations* of MultiMasher in Sect. 6. Conceptually, we have assessed MultiMasher along the dimensions of the logical framework for multi-device user interfaces presented in [7]. We have also taken into account further dimensions covering aspects specific to multi-device mashups, such as the type of components used or inter-component communication. For the technical evaluation, we tested MultiMasher on 50 popular web sites and, despite some limitations mostly affecting highly dynamic, JavaScript-heavy web sites, achieved encouraging results regarding its compatibility and support for browsing, distribution, and linking of web site elements.

2 Background

In [6], we define a multi-device mashup as "a web application that reuses content, presentation, and functionality provided by other web pages and that is distributed among multiple cooperating devices." To focus on the main concepts of a multi-device mashup, we introduce the following simple scenario (Fig. 1), which will be used as the running example in the paper.

> *Two friends, Bill and Ted, are on a holiday and want to plan a bike trip to several places nearby. While they each have their smartphone, they also want to make use of the larger, digital TV in their accommodation. As Wikipedia provides good background information, but does not give a good visual impression of a place, they want to simultaneously view Google Image Search results for each place they look up. Bill suggests that they could use MultiMasher to quickly mash up articles from Wikipedia and images from Google for the same place on the large screen, while they could each use their smartphone to enter new locations. However, they want to use a single input field which should update both the article and the images on the large screen rather than searching separately on each page. After collaboratively exploring a couple of places using their*

smartphones while sitting on the sofa, they later decide to move to Ted's laptop so that they can meet up with another friend to discuss their plans. Using MultiMasher, the mashup they created across the smartphones and the large screen can easily be migrated while preserving the current state.

Fig. 1. A multi-device mashup composed of two web sites and three devices.

This simple scenario illustrates the three areas of related work that our research builds on: mashups, DUIs and collaborative browsing.

The mashup in our scenario is spread across multiple devices and this distribution introduces a set of challenges, such as the migration of interface parts across devices, changing the distribution at run-time, and adaptation to device characteristics. Such problems have been addressed by DUI researchers and a number of frameworks have been built. Similar to our goals, MarcoFlow [8] aims at the composition of distributed mashup-like applications. Due to the high level of complexity, this approach is targeted at skilled developers and is not suitable for non-technical end-users. Other frameworks for DUI development follow a model-based approach, e.g. [9, 10]. A framework for the development of distributed interactive applications is introduced in [1]. It is divided into two components, a client side and engine side. Instead of relying on a fixed server, the user can flexibly configure any of the devices as the engine coordinating the distribution. The distributions can be updated at runtime, for example triggered by user interaction. Changing the distribution at runtime requires that interface components migrate from one device to another while preserving their state. This was the focus of their previous work [3], which allows the migration of existing web applications across devices. The system uses a proxy-based approach and uses DOM serialisation to propagate the state from one device to the other. Even though the system supports multiple users, its focus is on sequential interaction. It does not support the simultaneous interaction of multiple users or devices with a web application.

Another aspect in the scenario was the use of one input field to trigger a search in both web sites. The exchange of data and events between widgets is referred to as inter-widget communication (IWC). It is of interest to both the mashup and the DUI community, as it can be used to connect two widgets from different sources, but also to connect widgets across multiple devices. A recent

example from the DUI community is DireWolf [11], a framework for web applications based on pre-built widgets. It supports the distribution and migration of widgets at run-time. DireWolf implements both local IWC, as well as IWC across devices. In contrast to our work, a widget can only be present on one device at a time, while we support the replication of a component to multiple devices. In the mashup area, [12] presents a semi-automatic approach targeted at non-technical end-users to extend widgets with IWC capabilities through programming by demonstration. However, this is preliminary work that is still limited to only a few input scenarios. In the current version of MultiMasher, users have to explicitly link widgets from different sources, but their semi-automatic approach could be integrated with ours as they are both based on GUI-level events.

Besides IWC, component extraction is a topic that has been addressed by the mashup research community. In our scenario, components are extracted from existing web sites—Wikipedia and Google Image Search—and connected, so that communication between them is possible. Semi-automatic component extraction from existing web sites is, for example, supported by Firecrow [13]. The developer demonstrates to the tool the desired behaviour of the component by performing a series of interactions. The system tries to extract the necessary HTML, JavaScript, CSS and resources. The extracted UI controls may then be embedded into existing web sites. However, the tool does not provide support for the linking of multiple controls and any inter-widget communication must be implemented manually. On the other hand, mashArt [4] supports this using the approach of universal composition which allows the creation of applications based on the integration of data, application and UI components. The communication between the components is based on events and operations. However, the mashArt tool itself is targeted at advanced web users and the accompanying component library needs to be filled by professional programmers. In contrast, Ghiani et al. [5] present an environment for mashup creation targeted at end-users without programming knowledge. Similar to our work, mashup components can be chosen from arbitrary web sites through direct manipulation of the GUI. In order to connect mashup components, the system generates a list of input and output parameters by intercepting and analysing HTTP requests. The user may then associate input with output parameters across components.

As illustrated in the scenario, our system aims to support multiple simultaneous users: once web sites are distributed as components and mashed together, multiple users may interact with the same component in collaboration. We achieve this by integrating principles of collaborative browsing [14], or co-browsing, taking care of synchronising the browsers operated by multiple users. The issue of providing co-browsing of dynamic, JavaScript-enabled web pages is tackled in [15]. The presented solution works at the DOM level, can be implemented in JavaScript, and requires no extensions to the browser. The paper describes two mechanisms of JavaScript engine synchronisation. Input synchronisation takes into consideration UI events alone, thus synchronisation happens before the execution of JavaScript. Output synchronisation propagates the changes made on the DOM tree after the JavaScript execution. With MultiMasher, we

present a solution which uses similar concepts as those expressed in JavaScript engine output synchronisation. While the authors found input synchronisation to be better for scalability and user experience, it is not clear whether they take into account the fact that propagation of interactions may generate multiple updates to a database, for instance when submitting a form. To address this issue, we opted for an architecture that builds on a remote control metaphor similar to Hightlight [16], where interactions from mobile devices are executed on a proxy browser in the server and changes sent back to the mobile clients.

3 Concepts for Multi-Device Mashups

We have developed and integrated concepts for mashups, DUI and co-browsing that play an important role for multi-device mashups. In this section, we give an overview of the concepts in terms of entities and operations, before explaining how they were implemented in MultiMasher in the next sections.

Figure 2 illustrates the three main entities involved in a multi-device mashup: *components*, *mashups* and *devices*: Components extracted from existing web pages are composed into a set of inter-connected mashups which are accessed simultaneously by multiple devices.

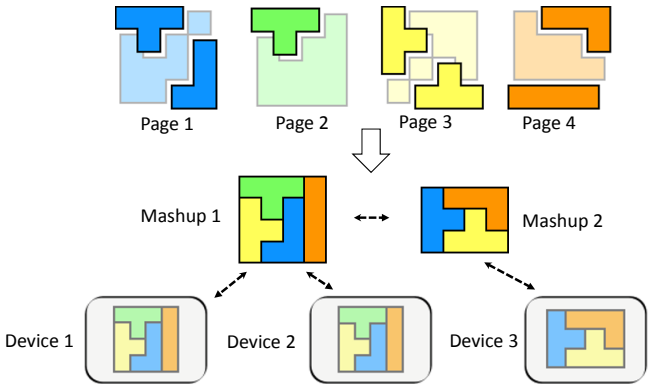

Fig. 2. Overview of a multi-device mashup

We define a component as a subset of web page elements, ranging from a single HTML element, such as a form input field, to the complete body. A mashup contains a specific set of components. All devices accessing the same mashup will receive the same set of components. In order to obtain a different view for another device, a mashup with a different set of components must be created. Note that there may be an overlap between the set of components used in two mashups. Finally, a multi-device mashup is defined as a set of mashups.

There should be no limitations to the kind of device that can access a multi-device mashup, granted that it is equipped with a modern web browser. The

relation between user and device may be one-to-one, many-to-one or one-to-many as, similar to our scenario, a user could use multiple devices at a time, for example a phone and a tablet, while a larger device such as a smart TV may be shared by multiple users.

In terms of operations, we distinguish three dimensions: *mashing up*, *distributing* and *co-browsing*. Mashing up relates to the operations that are also needed in traditional, single-device mashup development. *Component creation* should be supported programmatically or by direct manipulation of the web site. *Component manipulation and adaptation* operations, such as moving, resizing, copying or deleting, can provide a means for quickly building a visually appealing mashup. To obtain rich interactions, *inter-component communication* is required, so that interaction with one component may affect another component. We distinguish explicit and implicit inter-component communication (Fig. 3). Components that originate from the same web site communicate implicitly. For example, if one component displays the search bar of Google and another the list of results, entering a keyword in the search component should update the results component. As this is the default behaviour of the original web site, reproducing it in the mashup should not require any actions of the mashup developer. In contrast, components originating from two different sources must be explicitly connected if any interaction between them is required. Introducing a third component that contains the result of a search on Wikipedia into our example, it must be explicitly connected to the Google search component, if it is to update upon the input of a keyword into the Google search component.

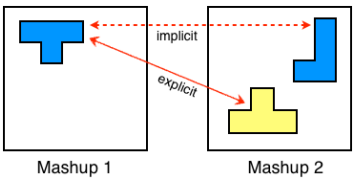

Fig. 3. Implicit and explicit inter-component communication

Distributing operations addresses the distribution of components across mashups and devices. To provide a means for experimentation, it should be possible to *migrate* components from one mashup to another (and consequently from one device to another), while maintaining its state and configuration, such as connections to other components. *Inter-component communication* should be transparent to the location of the participating component. That is, it should not make a difference whether two components reside in the same or in two different mashups. Note that two components inside the same mashup could also be accessed by multiple devices.

The co-browsing dimension covers the aspect of multiple users accessing a multi-device mashup simultaneously. As a user interacts with a multi-device mashup, updates to another user's view of the mashup are likely to occur. To

avoid confusion, a mechanism for *raising awareness for interactions* with the system should be employed.

4 MultiMasher

We have implemented the concepts described in the previous section in MultiMasher which provides visual tools for building and deploying multi-device mashups. It can be loaded into any modern web browser and does not require any extensions or plug-ins. As MultiMasher focuses on direct manipulation and requires no programming knowledge to build a multi-device mashup, our tool was designed to be used by technical and non-technical users alike. MultiMasher does not explicitly distinguish between design and run-time as any changes to a mashup immediately take effect. There are two main views in MultiMasher. The *global view* provides an overview of the complete multi-device mashup and all the mashups of which it is composed. Upon the selection of such a mashup, the *mashup view* is opened where the mashup can be edited and used.

4.1 Global View

The global view shows the state of the whole multi-device mashup and allows users to manipulate it. Initially, the user starts with an empty multi-device mashup. Any number of mashups can be created and added to the multi-device mashup, independent of the devices that are connected to the system. An interactive preview of all mashups is shown (Fig. 4). For each mashup, all the components it contains are colour coded. Components that originate from the same web site have the same background colour. The border colour denotes the set of web site elements that constitute the component. If two mashups contain the exact same component, it will have identical body and border colours in both previews. Components can be migrated between mashups via drag-and-drop and they can be resized directly in the browser. MultiMasher thus supports quick and easy experimentation and adaptation. Since all changes are immediately executed, all connected devices are updated simultaneously. For example, migrating a component from a source to a target mashup, removes the component from the view of all devices connected to the source mashup and introduces it on the target devices. To raise awareness, the global view lists all connected devices and for each mashup displays a colour-coded icon for each device subscribed to that mashup.

4.2 Mashup View

The mashup view displays a single mashup on a device. In this view, the user can interact with the mashup in the role of the end-user, e.g. entering a search query in our example. However, the mashup can also be edited. Components can be created and manipulated. Initially, a freshly created mashup provides an empty canvas. Clicking and dragging the mouse on the canvas specifies the dimensions

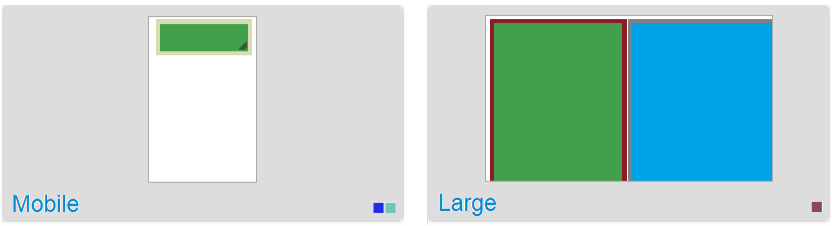

Fig. 4. An interactive preview of a multi-device mashup consisting of two mashups (Mobile and Large). The small coloured squares denote the clients connected to each mashup, two to Mobile, one to Large. The Large mashup contains two components from two different web sites. The Mobile mashup contains one component that originates from the same web site as the component of the left side of the Large mashup, which is indicated by the identical background colours.

of a new component. The system then creates a new component and asks the user to provide a source web site by either specifying a URL or choosing one that has been used previously from a list. MultiMasher loads the full web site inside the component boundaries. A component can be in either *execution mode* or *configuration mode*. In execution mode, the user interacts with the source web site of the component. In configuration mode, the user can adapt the component. Once in configuration mode, an *element selection mode* and a *linking mode* are available. During element selection, the user can select any element of the source web site. Subsequently, the component will only display the selected elements and its descendants.

As an example, in Fig. 5, only the main article of Wikipedia has been selected for a component. In the linking mode, users can add explicit inter-component communication. In the current version of MultiMasher, this can be done by attaching tags to change, submit and click events of a web site element. Subsequently, all elements with identical tags for the same event type will be connected and an event triggered on one element will trigger the same event on all connected elements. In our example, we can attach a *search* tag to both the Wikipedia and the Google search button for the click event, so that clicking the Wikipedia search button, also triggers the Google search button and vice versa. In configuration mode, the user can copy a component. This can be useful if the user wants to create another component based on the same web site or to have an exact copy of the component in another mashup. After copying the component in the mashup view, it can be migrated to another mashup in the global view. Also in the mashup view, components can be freely resized and moved to any location in the mashup.

4.3 Co-browsing Feedback

Since MultiMasher is targeted at multi-user scenarios, we added some features for co-browsing feedback to raise awareness for the interactions happening on all connected devices. In the global view, every connected client device is assigned

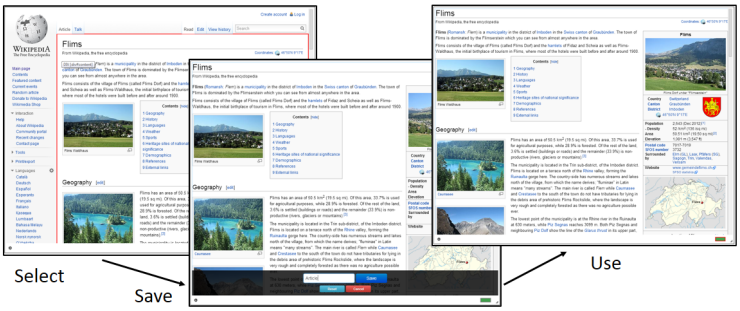

Fig. 5. Element selection mode.

a colour. Whenever that client interacts with a mashup or edits a mashup, all other clients are notified. For example, when a client moves a component inside a mashup, the component will be highlighted with that client's colour while it is being moved on all other devices. Furthermore, a notification message is displayed at the bottom corner of MultiMasher. When a user interacts with the web site inside a component, e.g. entering some text in an input field, the element will again be highlighted with the corresponding colour on all devices.

5 Architecture and Implementation

Our experiments with an early MultiMasher prototype [6] have shown that there is a need for an advanced architecture to support complex scenarios. A major challenge in an infrastructure for multi-device mashups is maintaining a consistent state across all involved devices. In our first prototype, we implemented a solution based on JavaScript engine input synchronisation as described in Sect. 2. DOM events originating from one device are replayed on all other devices. For example, when a user clicks on a button on one device, the click event is intercepted and sent to the server, which forwards it to all connected devices. On the devices, the click event is triggered on the button, thus, the remote click is replayed locally, which should result in the same state on all devices. There are two major drawbacks with this method. First, it can result in multiple updates to the server. For example, if a button to submit an order is clicked on one device, the same click will be repeated on all connected devices, possibly resulting in multiple orders. This is especially problematic in mashups that include purchases, e.g. in a mashup that combines eBay with a map. Second, depending on the server, different devices may receive different content, for example, a query may return different search results for two devices due to the search history, thus causing the system to run out of sync.

The root of the problem stems from having to maintain a consistent global state which is, however, fragmented and distributed across devices, each with its own session. Since we handle arbitrary external web sites which are outside our domain, a possible approach to this issue is to centralise the system: instead of

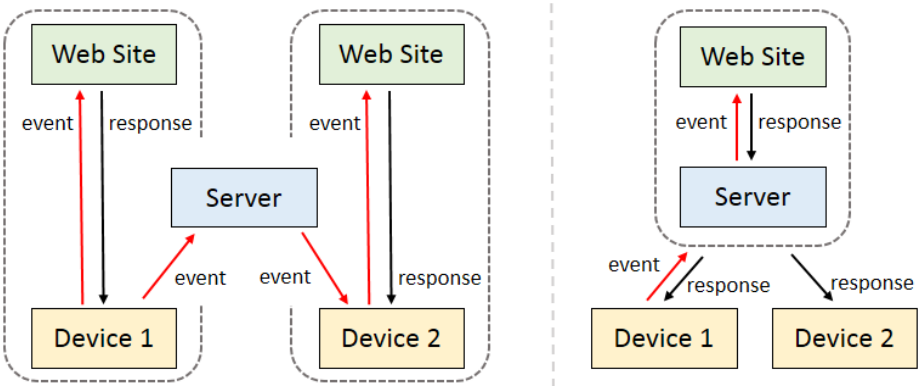

Fig. 6. Session synchronisation in the first prototype (left) and the current version (right) of MultiMasher.

trying to synchronise the state of multiple sessions that are distributed among devices, we propose to use a single session which is in turn shared across devices. To illustrate the difference, Fig. 6 compares our first prototype to our current approach. The former uses a server for broadcasting events to the devices, which have different sessions with the same web site, while the latter detaches the session from the devices moving it to the server, which becomes a proxy between the devices and the web site.

We propose an approach similar to the remote control metaphor used in Highlight [16]. On the server-side, a headless browser (i.e. a web browser without a GUI) maintains the state of each web site used in the multi-device mashup. Each source web site is loaded in a tab in the headless browser. All requests from client devices are proxied through the MultiMasher server, instead of directly going to the original web servers. For example, if a user clicks a button, the client device sends the click to the server which replays the click in the headless browser. This may lead to an update of the web site, e.g. by loading new content via Ajax or some local JavaScript manipulation. Subsequently, the MultiMasher server serialises the new DOM and sends it to all involved clients. To improve performance, resource requests, for example for images, are sent to the original web servers directly rather than being proxied.

On the client-side, each component representing part of a web site is contained in an iframe. By default, components are displayed in the size that they are received in the headless browser. This may not be suitable for small screens, but MultiMasher supports resizing of components which may be used to adjust the content for the devices in use. Inside the iframe, we disable the JavaScript of the original web site in order to gain full control over the component. All JavaScript is run in the headless browser on the server and the results are sent back to all clients. Thus we ensure a consistent state across all devices. For the selection of components in the client, we use a similar approach to [5]. However, instead of the element ID, we use its path in the DOM tree, thus avoiding the

explicit need for IDs. Yet, neither approach solves the problem of evolving web sites where both the DOM structure and element IDs may change.

We implemented MultiMasher using Node.js[1] on the server-side. As a headless browser, PhantomJS[2] is used, which runs in a separate process and communicates with Node.js via a plug-in.

6 Evaluation

In this section, we first present the results of a conceptual evaluation of MultiMasher along an established framework, followed by a technical evaluation based on 50 top web sites.

6.1 Conceptual Evaluation

In a conceptual evaluation, we assessed MultiMasher along the dimensions of a logical framework for multi-device UIs [7]. As this framework does not take into account aspects specific to mashups, we added the following set of additional dimensions that were derived during the process of building MultiMasher and from related work.

Inter-Component Communication describes how distributed UI elements can be set up to communicate with each other. This might be *automatic*, if no configuration by the user is required; *manual*, if the connection has to be manually established; and *mixed* if both cases are possible. MultiMasher provides a mixed approach. Automatic communication is only possible for components that originate from the same source, otherwise the communication has to be set up manually via tagging.

Synchronisation Consistency defines how complex it is to maintain synchronisation across devices and web sites. In a *consistent* system, resynchronisation is easy to achieve; while in an *inconsistent* system, resynchronisation is hard to achieve. An example of similar observations may be found in [15]. As MultiMasher stores the state of a mashup centrally on the server, a device that is out of sync simply needs to reconnect to the server in order to synchronise.

Type of Components analyses the type of components used in the distribution. These can be *widgets*, i.e. small, pre-built, self-contained web applications; or, as in MultiMasher, arbitrary *HTML elements*.

Component Creation defines the mechanisms that can be supported for creating components. Components may be *pre-built*, i.e. widgets; *scripted* at design-time, i.e. by developers working on the cross-device-mashup; or generated by *direct-manipulation* of web sites by the user as in MultiMasher.

[1] http://nodejs.org
[2] http://phantomjs.org

Flexibility to Changes analyses flexibility to changes in the source web sites, e.g. because of evolving web site structure of the re-authored pages [17]. It ranges from *high* if components can adapt well to the new source to *low* if no adaptation is provided. In MultiMasher, an evolving web site can interrupt inter-widget communication. If the DOM elements for a component can no longer be found, it only affects that component. The rest of the mashup remains stable and the user may adjust the component to any changes in the source web page.

Multi-Device Dimensions	MultiMasher	Mashup Dimensions	MultiMasher
Distribution	Dynamic	Inter-Comp. Comm.	Mixed
Migration	UI elements	Sync. Consistency	High
Granularity	Entire UI to components of UI elements	Type of Comp.	HTML element
Trigger	User	Creation	Direct manipulation
Sharing	Moving information, sharing by interacting	Flexibility	Medium
Timing	Mixed		
Modalities	Multi-modal		
Generation	Run-time		
Adaptation	Resizing		
Architecture	Client/Server		

Table 1. Conceptual evaluation for multi-device UI dimensions (left) and mashup dimensions (right)

We applied the original framework and the new dimensions to MultiMasher. A summary of the results can be found in Table 1. Compared to the other frameworks assessed in [7], MultiMasher is the only tool that supports multi-modal interaction. It can in principle be used with any interaction modality that is supported by the browser (e.g. touch). MultiMasher provides the most essential support for UI adaptation in that components can be moved and resized to accommodate different device characteristics. Advanced adaptation operations similar to CrowdAdapt [18] could be added in the future. Overall, our evaluation shows that MultiMasher can be well described in terms of the given dimensions and addresses some of the issues listed as future work in [7] such as preservation of state in UI migration.

6.2 Technical Evaluation

Based on the methodology used in [18], we evaluated MultiMasher in terms of its technical compatibility with 50 top web sites, ranked according to popularity and traffic by Alexa[3]. We selected the first 5 web sites from 10 categories: Arts,

[3] http://www.alexa.com

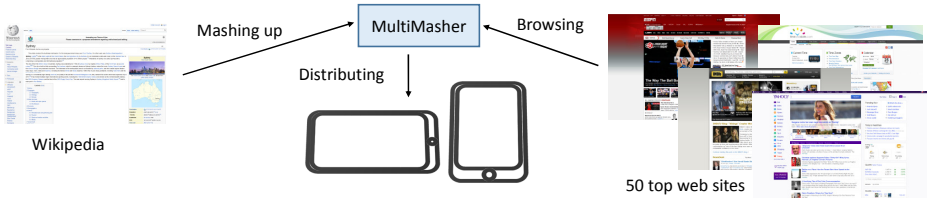

Fig. 7. Evaluation scenario.

Browsing	Page elements are loaded and displayed correctly. Page elements behave as expected. User interactions are handled correctly.
Distribution	Page elements can be extracted and distributed as components. Distributed elements are displayed correctly. Distributed elements behave as expected. Distributed elements are synchronised with the underlying web page.
Mashing up	Elements of a web page can be linked with elements belonging to another page. Click, change, and submit events are correctly replayed in linked elements.

Table 2. Criteria of the technical evaluation.

Business, Games, Health, Home, News, Science, Shopping, Society, and Sports. In order to assess MultiMasher with respect to these web sites, we developed a simple but representative cross-device mashup scenario (Fig. 7), which can be constructed using almost any type of web site. The scenario is composed of two devices, each using a different mashup, and two different inter-connected web sites. These are separated into components, which are then distributed and tested. In particular, we differentiate three categories of components: menu, search bar and main content. Only one of the two web sites is changed at each iteration of the scenario, while the other is fixed. The purpose of the fixed web site is exclusively to test the mashup capabilities (i.e. inter-page communication) of the cross-device mashup and it should not impact the evaluation. We selected Wikipedia, as it has proved to work very well in MultiMasher.

We assessed MultiMasher based on the criteria in Table 2 which can be grouped into three categories: browsing, distribution and mashing up. For each criterion, we assigned a value between *1 = poor* for web sites with major issues to *5 = excellent* for full support.

Overall the evaluation shows that MultiMasher offers good compatibility with web sites from many different domains. 43 of the 50 tested web sites (86%) obtained an average rating of a 4 or higher. Note that the browsing criteria have a higher priority than the other two categories. A low score in terms of browsing implies a low compatibility with MultiMasher overall, despite possibly higher scores in distributing and mashing up. Considering the browsing criteria in isolation, 31 web sites (60%) scored a 4 or higher. Thus, they either had

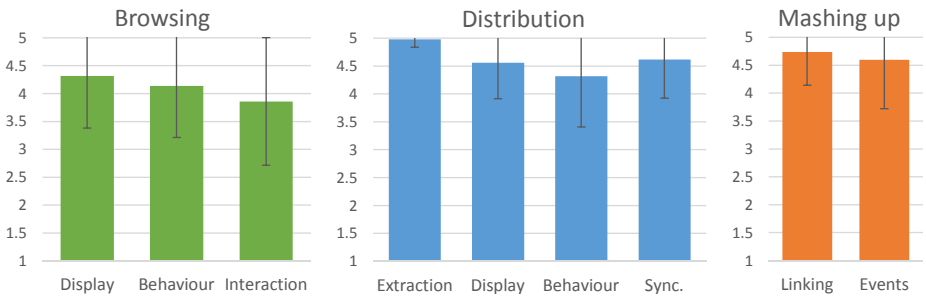

Fig. 8. Results of the technical evaluation (error bars showing standard deviation).

no issues at all or smaller issues affecting non-critical elements, such as ads in separate iframes. 48 sites (96%) scored a 3 or higher, which implies that the majority of critical elements were working as expected. MultiMasher showed very good compaitiblity with respect to the distribution and the mashing up categories. For the distribution criteria, 48 sites (96%) scored a 4 or higher and 17 sites were rated a 5. Similarly, for the mashing up criteria 44 sites (88%) reached at least a 4 and 39 sites (78%) obtainted the maximum rating of a 5.

Most issues we observed during the evaluation boil down to the following challenges:

- *Heavy use of JavaScript.* In order to prevent de-synchronisations and duplication of updates to the external web servers, we removed JavaScript from the local copy of the page in the front-end. This does not affect the normal behaviour of the page in most cases since JavaScript is run in the back-end, however, it limits cases where dynamic operations are expected to be available directly in the client, e.g. for drag-and-drop.
- *CSS extraction.* We encountered some instances where the extracted element was not positioned correctly inside the component or where the page background was shown out of place.
- *DOM evolution.* Certain web sites evolve their DOM, for instance by randomly changing form IDs to prevent spam. However, DOM evolution interferes with the path-based approach that MultiMasher uses to identify elements. DOM re-matching techniques, such as the ones introduced in PageTailor [17], could be used to alleviate this issue.

7 Conclusion

We have presented concepts for multi-device mashups and an implementation in MultiMasher based on an architecture with centralised state in a headless browser. Even though mashups and DUI could be viewed as two orthogonal concepts, we believe that the development of multi-device mashups benefits from an integrated solution, especially when considering an iterative approach to design and development. Future work could explore the potential of MultiMasher

to support multi-device development ranging from prototyping to testing and debugging of complex applications. While detailed user evaluations of MultiMasher remain as future work, we have paid particular attention to the design of its visual interface with the goal of allowing the quick and easy creation of user interfaces involving multiple devices.

References

1. Frosini, L., Manca, M., Paternò, F.: A Framework for the Development of Distributed Interactive Applications. In: Proc. EICS. (2013) 249–254
2. Melchior, J., Grolaux, D., Vanderdonckt, J., Roy, P.V.: A Toolkit for Peer-to-Peer Distributed User Interfaces: Concepts, Implementation, and Applications. In: Proc. EICS. (2009)
3. Ghiani, G., Paternò, F., Santoro, C.: Push and Pull of Web User Interfaces in Multi-Device Environments. In: Proc. AVI. (2012)
4. Daniel, F., Casati, F., Benatallah, B., Shan, M.: Hosted Universal Composition: Models, Languages and Infrastructure in mashArt. In: Proc. ER. (2009)
5. Ghiani, G., Paternò, F., Spano, L.D.: Creating Mashups by Direct Manipulation of Existing Web Applications. In: Proc. IS-EUD. (2011)
6. Husmann, M., Nebeling, M., Norrie, M.C.: MultiMasher: A Visual Tool for Multi-Device Mashups. In: Proc. ICWE. (2013)
7. Paternò, F., Santoro, C.: A Logical Framework for Multi-Device User Interfaces. In: Proc. EICS. (2012)
8. Daniel, F., Soi, S., Tranquillini, S., Casati, F., Heng, C., Yan, L.: Distributed Orchestration of User Interfaces. Inf. Syst (2012) 37(6):539556.
9. Melchior, J., Vanderdonckt, J., Roy, P.: A Model-Based Approach for Distributed User Interfaces. In: Proc. EICS. (2011)
10. Paternò, F., Santoro, C., Spano, L.D.: Maria: A Universal, Declarative, Multiple Abstraction-Level Language for Service-Oriented Applications in Ubiquitous Environments. In: TOCHI, 16(4). (2009)
11. Kovachev, D., Renzel, D., Nicolaescu, P., Klamma, R.: DireWolf - Distributing and Migrating User Interfaces for Widget-Based Web Applications. In: Proc. ICWE. (2013)
12. Chudnovskyy, O., Fischer, C., Gaedke, M., Pietschmann, S.: Inter-Widget Communication by Demonstration in User Interface Mashups. In: Proc. ICWE. (2013)
13. Maras, J., Stula, M., Carlson, J.: Reusing Web Application User-Interface Controls. In: Proc. ICWE. (2011)
14. Greenberg, S., Roseman, M.: GroupWeb: A WWW Browser as Real Time Groupware. In: Proc. CHI. (1996)
15. Lowet, D., Goergen, D.: Co-browsing Dynamic Web Pages. In: Proc. WWW. (2009)
16. Nichols, J., Hua, Z., Barton, J.: Highlight: A System for Creating and Deploying Mobile Web Applications. In: Proc. UIST. (2008)
17. Bila, N., Ronda, T., Mohomed, I., Truong, K.N., de Lara, E.: PageTailor: Reusable End-User Customization for the Mobile Web. In: Proc. MobiSys. (2007)
18. Nebeling, M., Speicher, M., Norrie, M.C.: CrowdAdapt: Enabling Crowdsourced Web Page Adaptation for Individual Viewing Conditions and Preferences. In: Proc. EICS. (2013)